An Implementation of CCSDS 122.0-B-1 Recommended Standard

Mar. 18, 2008

Hongqiang Wang.

1. Disclaimer

(1) Before you download and use the program, you must read the license file and accept the terms and conditions.

(2) We provide the source code and the software WITHOUT ANY WARRANTIES. The users will be responsible for any lose or damage caused by the use of the source code and the software.

2. Author and contact information

Hongqiang Wang
PhD candidate & Research assistant
Department of Electrical Engineering
University of Nebraska-Lincoln
Tel: 402-472-1973
Fax: 402-472-4732
Email: hqwang@bigred.unl.edu, hqwang@eecomm.unl.edu

Comments and suggestions are welcome. Please send me emails if you have questions or find bugs. I would try my best to answer and greatly appreciate your efforts to make this program better.

3. Acknowledgment

This project has been supported by a funding from NASA, and the Department of Electrical Engineering, the University of Nebraska-Lincoln (UNL). The development and test of the software would not be successful without the help of Dr. Pen-Shu Yeh at NASA. We have been working very closely throughout the software test process. I am grateful for her great patience and thorough guidance. I also appreciate my advisor Dr. Sayood Khalid for his encouragement and guidance, especially in the early stage of this project. Thanks also go to Dr. Mark Bauer, who helped us set up and configure the web server to make the website work nicely.

4. Compression Program Description

We present some basic information on the implementation of the data compression scheme. This program is developed based on the Blue Book 122.0-B.1 by the CCSDS released on March 31, 2006. For more details on the compression recommendation, please visit www.ccsds.org. For more details on the program, please look into the source code that is downloadable at http://hyperspectral.unl.edu. Users can run and test the code from the web server directly. Or, users can download source code and executable code, and test by themselves.

The program is developed and tested with Microsoft Visual C++ 2005 in Windows XP. It has been tested under Linux Fedora 5.0 as well.

4.1. Files

The program contains 3 header files, and 18 C files.

Three header files:

getopt.h:

This file is obtained for free under GNU General Public License

tailor.h:

This file is obtained for free under GNU General Public License

These two files are the header files for getopt.c. getopt.c is to extract the coding parameters from the string in the command line. So basically these two files do not have direct link with the compression.

global.h:

This file defines the macro variables, structure, and types that are globally used in the coder. Some global functions are also defined.

The C files have many functions defined, which will be listed in later section. So here we do not list the files.

4.2. Structures and Union

The definitions of structures are in bold capital letters.

SYMBOLDETAILS

This records the value, mapping value, length, sign, and type of a symbol. This is used for the bit plane coding.

BITSTREAM

This structure contains the information of bit stream, including the total number of bits in current segment, the total number of bits output so far; byte buffer; and the number of bits in a byte that have filled, and FILE type for bit stream output. This one is used throughout the encoding and decoding process.

TYPEC

Record the type of children subblock.

TRANH

Record the transition to grandchildren subblock.

TRANHI

Record the transition to grandchildren subblock Hi.

TYPEHIJ

Record the type of grandchildren subblock Hij.

PARENTREFINE

Record the refine bits of parent subblock.

CHILDRENREF

Record the refine bits of children subblock.

GRANDCHILDREDREF

Record the refine bits of children subblock.

REFINEMENTBIT

Record all refine bits in a block, i.e., including PARENTREFINE, CHILDRENREF, and GRANDCHILDREDREF.

PLANEHIT

Record all type bits and transitional history.

BLOCKBITSHIT

Record all coding information of a block, including DC, AC, mapping, maximum AC depth, and some coding structures defined above.

HEADER_STRUCTURE_PART1

Part 1 structure of header structure.

HEADER_STRUCTURE_PART2

Part 2 structure of header structure.

HEADER_STRUCTURE_PART3

Part 3 structure of header structure.

HEADER_STRUCTURE_PART4

Part 4 structure of header structure.

HEADER

Consists of the above 4 header structures.

STR_STOPLOCATION

Structure to record the partial decoding locations. Used for decoding only.

CODINGPARAMETERS

Contains all coding parameters, image information, files to output and input, rate to be reached.

BLOCKSTRING

For decoding only. A link will be built to record the decoded blocks. The link is reorganized and reconstructed using inverse wavelet transform (IWT).

There is one UNION type:

HEADERUNION

This is to save storage space and facilitate some operations.

4.3. Functions

ACBpeDecoding

AC component decoding. Called by DecoderEngine.

ACBpeEncoding

AC component decoding. Called by EncoderEngine.

ACDepthEncoder

ACdepth encoding. Called by ACBpeEncoding.

ACDepthDecoder

ACdepth decoding. Called by ACBpeDecoding.

ACGaggleEncoding

This is for AC encoding of gaggles. Called by ACDepthEncoder.

ACGaggleDecoding

This is for AC decoding of gaggles. Called by ACDepthDecoder.

AdjustOutPut

Called by DecoderEngine(StructCodingPara *PtrCoding).

BitPlaneSymbolReset

Reset the symbols in the bitplane structure. Called by StagesEnCodingGaggles1, StagesEnCodingGaggles2, and StagesEnCodingGaggles3.

BitsOutput

Outputs bits to buffer and files. For encoding only.

BitsRead

Read bits to buffer and files. For decoding only.

BlockScanEncode

Kernel part of the bit plane coding. It determines the type of subblocks and transitional bits. Called by ACBpeEncoding only.

BuildBlockString

This is to build block string index for further encoding. Called by EncoderEngine only.

CheckUsefill

This is to check if UseFill is enabled, and if so, how many bits need to fill. Called by ACBpeDecoding only.

CodingOptions

This is to determine the coding parameters of Golomb-Rice decoding. Called by StagesEnCoding only.

CoeffDegroup

Reorganize back into the block ready for inverse wavelet transform. Called by DecodingOutputInteger only.

CoeffDegroupFloating

Reorganize back into the block ready for inverse wavelet transform. called by DecodingOutputFloating only.

CoefficientsRescaling

Rescale the coefficients before inverse DWT, called by DWT_Reverse only.

CoefficientsScaling

Rescale the coefficients after DWT, called by DWT_ only.

CoeffRegroup

Reorganize the transform coefficients into the blocks after wavelet transform. Called by DWT_ only.

CoeffRegroupF97

Reorganize the transform coefficients (float) into the blocks after wavelet transform. Called by DWT_ only.

ConvTwosComp

DC into 2's complement representation. Called by DCEncoding only.

DCDeCoding

DC decoding, called by DecoderEngine only.

DCEncoder

DC entropy encoding, called by DCEntropyEncoder only.

DCEncoding

DC encoding, called by EncoderEngine only.

DCEntropyDecoder

DC entropy decoding, called by DCDeCoding only.

DCEntropyEncoder

DC entropy encoding, called by DCEncoding only.

DCGaggleDecoding

DC decoding in a gaggle, called by DCEntropyDecoder.

DebugInfo

Output debug information.

DecoderEngine

Kernel function for decoding, called by main only.

DecodingOutputFloating

Output the floating point DWT decoding results, called by DecoderEngine only.

DecodingOutputInteger

Output the integer DWT decoding results, called by DecoderEngine only.

DeConvTwosComp:

Convert coefficients back from 2 s complement representation to normal, called byAdjustOutPut only.

DeMappingPattern:

Demap the decoded symbols back to its original value. Called by StagesDeCodingGaggles1, StagesDeCodingGaggles2, StagesDeCodingGaggles3

DPCM_ACDeMapper

Demap the decoded ACdepth back to its original value. Called by ACDepthDecoder.

DPCM_ACMapper

Map the ACdepth from double (negative and positive) side value to positive value. Called by ACDepthEncoder.

DPCM_DCDeMapper

Demap the decoded DC value back to its original value. Called by DCDeCoding.

DPCM_DCMapper

Map DC from double (negative and positive) side value to positive value. Called by DCEncoding.

DWT_

DWT transform, called by EncoderEngine only.

DWT_Reverse

Inverse DWT transform, taking integer input,called by DecoderEngine only.

DWT_ReverseFloating

Inverse DWT transform, taking floating input, called by DecoderEngine only.

EncoderEngine

The kernel function for decoding, called by main only.

ErrorMsg

This is to output error code to a file and standard terminal.

forwardf97f

Forward transform of floating 97.

forwardf97M

Forward transform of integer 97.

HeaderInitialize

Initialize the header before coding. Called by main only.

HeaderOutput

Output the header. For each segment, header will be output. Called by DCEncoding.

HeaderReadin

Read the header from the coded bit stream. Called by DecoderEngine.

HeaderUpdate

Header is updated after the coding of a segment and ready for the coding of next segment called by EncoderEngine only.

ImageRead

Open an image to read. Called by EncoderEngine.

ImageSize

Determine the image size. Called by EncoderEngine.

ImageWrite

Output the decoded image. Called by DecodingOutputInteger.

ImageWriteFloat

Output the decoded image (in floating mode) DecodingOutputFloating(StructCodingPara *PtrCP, float **imgout_floatingcase).

inversef97f

inverse floating DWT transform.

inversef97M

inverse integer DWT transform.

lifting_f97_2D

lifting algorithm of floating 97 DWT transform.

lifting_M97_2D

lifting algorithm of integer 97 DWT transform.

main:

main function.

OutputCodeWord

Output a codeword. Called by BitsOutput (StructCodingPara *Ptr, DWORD bit, int length).

ParameterValidCheck

Called by main to verify the coding parameters are valid.

PatternMapping

Map the original symbol to new one according to table. Called by CodingOptions.

RefBitsDe

Get refine bits from the coded bit stream, called by StagesDeCoding.

RefBitsEn

Output refine bits from the coded bit stream, called by StagesDeCoding.

RiceCoding

Rice encoding based on the coding parameters. Called by StagesEnCodingGaggles1, StagesEnCodingGaggles2, and StagesEnCodingGaggles3.

RiceDecoding

Rice decoding based on the coding parameters. Called by StagesDeCodingGaggles1, StagesDeCodingGaggles2, and StagesDeCodingGaggles3.

SegmentBufferFlushDecoder

Flush the leftover bits in a segment coding and reset the byte alignment for the coding of next segment. Called by EncoderEngine.

SegmentBufferFlushEncoder

Flush the leftover bits in a segment decoding and reset the byte alignment for the decoding of next segment. Called by DecoderEngine.

StagesDeCoding

Perform the three stages bit plane decoding.

ACBpeDecoding

AC decoding.

StagesDeCodingGaggles1

Bit plane Decoding of stage 1, Called by StagesDeCoding.

StagesDeCodingGaggles2

Bit plane Decoding of stage 2, Called by StagesDeCoding.

StagesDeCodingGaggles3

Bit plane Decoding of stage 3, Called by StagesDeCoding.

StagesEnCodingGaggles1

Bit plane encoding of stage 1, Called by StagesEnCoding.

StagesEnCodingGaggles2

Bit plane encoding of stage 2, Called by StagesEnCoding.

StagesEnCodingGaggles3

Bit plane encoding of stage 3, Called by StagesEnCoding.

TempCoeffOutput

Temporal coefficients output, for debug purpose.

4.4. Coding Parameters in Command Line

If the source files have been compiled and the executable file is generated, the program can be run as follows (assume the executable file is called ***bpe)***:

- ***bpe [-e InputImageName] [-d CompressedFileName] (-o OutputFileName) [-r BitsPerPixel] [-w ImageRows] [-h ImageColumns] [-b BitsPerPixel] [-f ByteOrder] [-g UnSigned] [-t TypeOfDWT] [-s BlocksInSegment]***

Note that the parameters are not case-sensitive. **[ ]** means the parameters are optional, and **( )** means they are mandatory. The order of these parameters is arbitrary.

Parameters:

- **[-e *InputImageName*]:** Take a raw image file called ***InputImageName*** for encoding. (for encoding only)

- **[-d *CompressedFileName*]:** Take a compressed file called ***CompressedFileName*** for decoding. For decoding only. Note that **[–e]** cannot not be used with this one at the same time. (for decoding only)

  Note that one of above two parameters must be provided.

- **(-o *OutputFileName*):** Output a file called ***OutputFileName***. It takes character string. If this is used with **[–e],** filename is for the compressed bitstream. If this is used with **[-d],** then output the decoded image. (This is for both encoding and decoding, and mandatory)

- **[-r *BitsPerPixel*]:** desired compression ratio (bits/pixel). This parameter is valid for encoding and some decoding scenarios.
  1) For encoding, the encoder will compress the image to this rate if possible. There are two extreme conditions to be clarified: If the ratio specified is too small, error will be reported. A valid ratio depends on the header, segment size, etc. For example, if the

ratio is so small that the header has not been fully output, coding error will be reported. If the ratio is too large, i.e., the lowest bit plane has been coded but ratio is still not achieved, the coding may stop, or fill some 1s, dependent on the parameter **UseFill** in the header.

2) For decoding, this parameter is for embedded decoding. If the ratio is greater than the actual ratio of the image coded, the decoder will decode to the actual ratio. If the ratio is less than the ratio of encoding, the decoder will work until reaching this ratio and then discard the rest bits in the current coding segment. And decoder then continues to decode the next segment, as long as the decoder knows where the next segment starts. That is, for success of embedded decoding, the decoder needs to know the packet size, which can be defined in the header. If **SegByteLimit** in the header is non-zero and **UseFill** is true, the decoder knows the exact location in the bit stream of each segment and embedded decoding can be carried out.

- **[-b *OriginalBitsPerPixel*]:** the number of bit of a pixel. By default it is 8. We can only handle up to 16 bits/pixel. So this parameter must be less than or equal to 16. (for encoding only)

- **[-w *ImageRows*]:** the number of pixels in a row. (for encoding only)

- **[-h *ImageColumns*]:** the number of pixels in a column. (for encoding only)

Ideally, the number of rows and columns are integer multiples of 8. This would facilitate the wavelet transform and block grouping. However, this requirement is not mandatory.

This program could handle odd-sized raw images, i.e., the number of rows and columns of the raw image are not necessarily integer multiples of 8. If they are not, we simply replicate the last row or the last column until they are integer multiples of 8.

These two parameters along with the bytes/pixel will be used to verify if the image size is correct. If the ***ImageRows*** * ***ImageColumns*** * ***OriginalBitsPerPixel*** is not the same as the image size, error will be reported.

These parameters are for encoding only. For decoding, the coder can determine the image size by examining the information in header and therefore it is not necessary to specify the image dimension.

- **[-f ByteOrder]:** byte order of a pixel, taking 0 or 1. If a pixel consists of 2 bytes, we need the exact byte order. 0 means litttle endian, 1 means big endian. Default value is 1. If bits/pixel of the image is less than 8, this option is not necessary (for encoding only).

- ***[-g Signed]*** The pixels are unsigned or signed. By default it is unsigned. If 1, pixels are signed. Or it is unsigned.

- **[-t *TypeOfDWT*]:** type of discrete wavelet transform, taking 0 or 1. 1 represents integer 9-7 DWT and 0 represents floating 9-7 DWT. By default, coder takes integer 9-7 DWT. (for encoding only)

- **[-s *BlocksInSegment*]:** the number of blocks in each segment. Default value is 256. This value has to be greater than or equal to 16. If it is less than 16, error may occur. And this value should not exceed the maximum number of blocks of the image. (for encoding only)

Example 1: ***bpe -e sensin.img -o codes -r 1.0 –w 256 –h 256 –s 64 –b 8 –t 0***

This is to encode a raw image called ***sensin.img*** (256 X 256, 8bits/pixel, unsigned) with floating 9-7 DWT, 64 blocks in segment, a desired compression ratio of 1.0bits/pixel, and an output filename called ***codes***.

Example 2: ***bpe -d codes -o ss.img***

This is to decode a compressed file called ***codes,*** and output the reconstructed raw image into a file called ***ss.img***.

4.5. Other Coding Parameters

The compression recommendation defines some coding parameters that users can adjust to accommodate their applications. The parameters in the command line are critical. However, to take advantage of its flexibility, it is better to check the parameters that are defined in header (see function ***HeaderInitialize)***. This function is to initialize coding parameters. Note that not all coding parameters can be adjusted by users. A portion of them depend on the coding process. Some of them can be specified by users, but they cannot be changed once coding starts, such as the coding parameters in header 4. For the parameters that users can modify, please refer to the recommendation. For reference, we build tables to list the parameters and their default (or initial) values in our implementation. Note that all reserved bits are set to 0.

Table 1: fields in header part 1 and their default (or initial) values

Field	Default value (or initial value)	Description
StartImgFlag	1	Flag this is the first segment. After first segment, this will be turned to 0
EndImgFlag	0	Flag this is not the last segment. Once the coder determines this is the last segment, this will be turned to 1
SegmentCount	0	Segment count value (mod 256), from 0 to 255.

BitDepthDC	0	This will be updated by coder after the coder analyzes the DCs in the segment.
BitDepthAC	0	This will be updated by coder after the coder analyzes the ACs in the segment.
Part2Flag	1	Indicate Header part 2 will be present
Part3Flag	1	Indicate Header part 3 will be present
Part4Flag	1	Indicate Header part 4 will be present
PadRows	0	This will be present only when EndImgFlag becomes 1.

In the first segment, Part2Flag, Part3Flag, and Part4Flag are initialized to 1. After the first segment, these 3 parameters are flagged to 0 (see function: ***headerupdate***) in our implementation. Note that usually head part 4 is not changeable in the entire coding process. Therefore Part4Flag should be 0 after first segment.

Table 2: fields in header part 2 and their default (or initial) values

Parameter	Default value	Description
SegByteLimit	0	No byte limit in segment.
DCStop	0	Will not stop coding after DC is coded. If this is 1, coding of this segment stops once DC is coded.
BitPlaneStop	0	The bit plane at which the coder will code and then stop. If this is 0, coder will complete coding of all bit planes.
StageStop	3	The stage in which that the coder will code and then stop. If this is 3, then coder will complete all stages.
UseFill	0	If SegByteLimit is 0, this must be 0. Otherwise, this can be 0 or 1.

Table 3: fields in header part 3 and their default (or initial) values

Parameter	Default value	Description
S	256	256 blocks in a segment
OptDCSelect	1	Use the optimized method to choose the coding parameter for coding of DC.
OptACSelect	1	Use the optimized method to choose the coding parameter for coding of AC.

Table 4: fields in header part 4 and their default (or initial) values

Parameter	Default value	Description

DWTtype	1	Integer 9-7 DWT, if 0 floating DWT
PixelBitDepth	8	8 Bits/pixel. If this is 0, 16 Bits/pixel
ImageWidth	2048	This will be updated from command line
TransposeImg	0	Not transpose
SignedPixels	0	Unsigned.
CodeWordLength	0	8-bit word
CustomWtFlag	0	No custom weighting factor
CustomWtHH$_1$		
CustomWtHL1	1	These are the default custom weighting factors for different sub bands. If CustomWtFlag is 0, all these will be set to 0.
CustomWtLH1		
CustomWtHH2		
CustomWtHL2	2	
CustomWtLH2		
CustomWtHH3	3	
CustomWtHL3		
CustomWtLH3		

4.6. Quality-mode

The coder can work in quality-mode so that users can specify whether the coding stops after DCs are done, or at certain bit planes, or in some particular stage stages. The quality-mode provides a great flexibility to meet the needs of various applications.

The parameters defined in the header part 2. DCStop, BitPlaneStop, and StageStop in Table 2 can be adjusted to make the coder run under quality-mode. All these parameters can be modified in header.c. For each segment to be coded, the three parameters work in the way as follows:

- If DCstop is set to 1, then only DCs are coded, and all bit planes in the segment will be ignored. If it is set to 0, coding of this segment will continue after DC is coded.
- By setting BitPlaneStop to nonzero value, the coder will stop coding after BitPlaneStopth bit plane is coded. BitPlaneStop occupies 5 bits so that user can specify up to 31 individual bit planes.
- For StageStop (2 bits), the coder will stop if the StageStop stage is done. If StageStop is 3, all stages will be coded. There are 4 stages totally and so user can specify any stage other than 3 to terminate the stage coding earlier.